Project plan

Kent Wincent Holt - 473209

Eldar Hauge Torkelsen - 473180

Zohaib Butt - 473219

January 2019

Contents

List of Figures

List of Tables

1 Project goals and constraints

1.1 Background

This project was requested by Frantz, Christopher (product owner) from the Department of Computer Science at NTNU. The project was requested because finding libraries or frameworks that would fit well for new software projects can be a time consuming and difficult process that might require white-box analysis. This project should make the white-box analysis easier by abstracting the code and represent it as a 3D graphical structure and giving information on complexity and quality of the project.

CodePark [1] and CodeCity [2] are two applications trying to solve the same type of problem with similar 3D solution, representing the code as houses or city blocks. CodePark visualize the code base in park-like environment. Each class in the code base is represented as a interactable cube. In the cube, all functions in each class are visible on the wall and upon a click, it highlights the function in the code, which is shown on a separate wall. CodeCity represents the code as navigable cities, where each building represents a class and each district represents a package. The two examples differ from this project in how this project will represent connections related to execution flow and the code not being represented as buildings or cities. This project will also represent different data structures such as vectors and maps.

1.2 Project Goal

We have divided our goal in three sections result goals, effect goals and learning outcomes. Result goals defines what we deliver as the end product. Effect goals describe what the team expect from the system after a set amount of time and are listed to give an indication of the quality the developers should aim for. Learning outcomes describe what the team wants to learning during development and thesis writing.

1.2.1 Result Goal

- Program must, at a minimum, be able to abstract and visualize language version up to Java 11.0.2 and C++17.

- Code visualization must be in 3D.
- Program must give an indication of the quality of the software code.
- Program must be able to take a link to a git repository to use as basis for visualization.
- Program must be able to show potential execution path or program flow through the program.
- Program must have a public facing API for developers.
- Program must be dockerized.
- The user must be able to interact with the 3D visualization to show implementation of data structures.
- The system must present complexity metrics to the user.
- The system must give an indication on where complexity arises.

1.2.2 Effect Goal

1. Quantitative goals for the system:
   - The system must handle 30 concurrent users with repository of less than 10 KLOC while using less than 10 minutes of processing time.
   - The system must be able to handle a peak of at least 60 users.
   - The system must be able to handle repositories of less than 100 KLOC.
2. Qualitative goals for the system:
   - The system must help developer(s) gain overview and an understanding of the code base.
   - The system must let the user easily navigate through the visualization.
   - The system must be naturally intuitive and easy for new users.
   - The system must be accessible to people with minor disabilities like colorblindness.

1.2.3 Learning outcome

- Learn in depths WebGL technology.
- Learn professionalism in web development.
- Understand and follow DevOps methodology through out the project.
- Learn in depths dockerizing containers.
- Learn Front-end web frameworks.
- Learn about different code quality metrics for static analysis.
- Learn about professional web architectures.
- Learn about Infrastructure as Code.

1.3 Constraints

1.3.1 Legal Constraints

- The resulting system will not be in violation of general data protection regulation. [3]
- The system will not be in violation of Norwegian copyright act.

1.3.2 Technological Constraints

- The system should be cross platform.
- The system should be dockerized.
- The system should have a REST API.

1.3.3 Time Constraints

- The system should be finalized by 16^{th} May.

2 Scope

2.1 Subject area

Our software is meant to be used in a software development setting. Software development includes many different processes such as system specification, software reuse, integration, development, testing and maintenance. Our system will mostly help with the software reuse and integration part, but will also indirectly affect the other processes.

System specification deals with identifying what the system should do and the environment it acts in.

Software reuse takes those specifications and identifies preexisting systems that could be used instead of developing new software or could be used to make development easier. This process often identifies libraries and frameworks that can be used. Using libraries or frameworks in the systems is often recommended [4] in both large and small systems. Developers spend a lot of time debugging, securing, documenting, structuring, finding suitable libraries, finding frameworks, etc. Finding libraries and frameworks to be used in a project can be frustrating and time consuming if there is no good documentation, source code or life cycle provided [5].

Software integration takes the preexisting systems and either mold them or the software it integrates with to fit smoothly. The integration can vary in difficulty depending on the implementation of each software solution.

Software development creates the functionality that is not provided by preexisting solutions. It is recommended to have as little code complexity [6] as possible for readability [7] which can be hard for developers to see while programming and is partially reliant on how well the software integration went. Potential problems with the preexisting systems are likely to emerge at this state as it is difficult to see limitation and bugs during the software integration phase. Finding bugs in this scenario is time-consuming [8] and might require the developers to do white-box testing on an unfamiliar system. Bugs that are not found during development will hopefully be found during software testing.

Software testing validates that the functionality works as expected with no side-effects. This can help spot bugs or shortcomings in the preexisting systems. If bugs are not found during software testing, then they are likely to persist through the life cycle of the software or until identified by the user

Software maintenance deals with updating the software to remove issues found by users or through normal use. It also assures that dependencies stay up to date.

The problem our system will help with is giving an impression of the build and quality of libraries and frameworks that are considered by the software reuse process as a potential addition for their project. The system will act as a lesser alternative to documentation or as a quick overview to the libraries or framework.

2.2 Boundaries

The system is not meant to help with software specification or the later stages of development after software integration. It is not meant to supersede or be an exhaustive substitution of documentation, but rather an abstraction and quick overview of the code base. The abstraction will be limited to a 3D visualization and textual representation of metric and meta data. It will not change the software in question or improve the quality, but give the end user an insight into the software model and code quality. The project will not involve developing new quality metrics, but use preexisting metrics that have been used by other developers and been proven to be help-full.

The system will not build, nor execute the code base in question but rather do static analysis on the code base. There will neither be provided any test coverage nor execution of any shipped testing functionality.

2.3 Task description

The application should assist in getting an overview over the structure and complexity of the product in question.

The program should take in a Git repository link and/or code base for analysis. The analysis must contain information about code such as lines of code, functions, classes, objects, templates, name-spaces, tree structures, multidimensional maps and relationships between structures. The gathered information should then be abstracted to give relevant and standard quality metrics whilst also showing a 3D representation of the program.

The 3D representation must be understandable and relevant for a software developer. As this is not a well defined metric, targeted user testings must be performed to verify that this criteria is met and must indicate that the system is user friendly and helpful. The metrics to be shown and form of visualization should be determined by the development team through the development process.

The 3D representation should include name of functions and structures for ease of localizing the structure relative to others.

The 3D visualization should be navigable by the end user. The navigation should be intuitive and able to get to important aspects of the program. Navigation should include free camera movement, on track movement and jump/teleport. Track movement should be able to follow connections or relations from function or class, without ending up off target. Jump/teleport movement should allow user to click and get to data structures and code, which can be seen from the users navigational position.

The 3D visualization should also be interactive, letting the user collapse and expand functions, classes and other data structures to show more or less information about them. Information that the user should be able to hide or show including code implementation and metadata. Clicking a data structure should highlight connected data structures. Hover over a data structure should show comments connected to the data structure.

The program should display common quality metrics like coupling and cohesion. The metrics should include ideas from cyclomatic complexity and, where sensible, connascence metrics. Based on the quality metrics, the program should give an estimate of code quality.

Program modularity and scalability is preferred. If possible and time allows, then the program should use docker technology.

The program should be cross platform and requires very little or no installation for end users. It is expected that this will be solved by making it a web application but this is not a requirement.

3 Project organization

3.1 Responsibilities and roles

Group roles	
Member	Role
Kent Wincent Holt - 473209	The Group leader The lead programmer Software developer/tester The release manager
Eldar Hauge Torkelsen - 473180	DevOps evangelist Scrum master The automation architect Software developer/tester Lead security engineer Recorder The Lead Interaction designer
Zohaib Butt - 473219	Software developer/tester The experience assurance (XA) professional Facilitator The utility technology player

Table 1: Group roles

3.2 Routine's and group rules

All group members are expected to work from Monday to Friday from 08.15 to 16.00. If the group sees it necessary to work overtime then the group should work past 16.00 and on Saturdays.

From Monday to Friday, lunch breaks are expected to take less than 45 minuets and the timing is up to the individual and and breaks should be taken when the individual member feels that a break would increase efficiency in the long term or a break is necessary. Sunday is considered a day off, but if a group member wants to work voluntarily, they are allowed, but can't make any claims/excuses related to the work done. Voluntary work should be tracked and labeled as such. Members are encouraged to use Sunday as a break day.

When possible, meetings should be recorded and a meeting report should be created.

If a group member lose 3 days of work without valid reasoning, then a warning will be given. If the warning is unheeded the supervisor will be included and will decide on further actions.

4 Planning, followup and reporting

4.1 Development process

4.1.1 Project characteristics affecting choice of software development process

The system needs to be developed in a short amount of time and the system specifications are partially ambiguous, requiring refinement during development. This is an effect of the research focused topic and would benefit from an agile development method. The project can be considered a success even if it lacks minor features so a development method allowing for iterative improvements would be beneficial. The product owner has given the impression that it would be nice with a prototype version that could be used before the final version.

The development team consists of three persons with only moderate experience so a methodology allowing for low boundaries for communication and close teamwork would work well.

4.1.2 Choice of software development process

Due to the ambiguous specification of the project paired with the fact that changes will take place. Agile methodologies gives us the flexibility needed for the project. Weekly meetings with product owner and user studies will continuously affect the project. As a team we also want to have fixed routines and structure. Therefor the team has decided to use Scrum as a system development process along with DevOps for continuous integration

To have good documentation we will automate documentation using APIDOC, GoDoc and JSDoc tools. For further documentation we also focus on ideas like Infrastructure as Code (IaC) for documenting architecture and environment. Using these tools and IaC along with scrum requirements for regular meetings and reporting and DevOps gives us a good platform to conduct project in agile but also in a structured manner.

The team will setup backlog with user stories which are acquired from user studies. The team has planned to have weekly sprints. Sprint will start on Mondays and end on Fridays. Longer sprint length for a bachelor project does not seem feasible since this will give less opportunities for the product owner to give his input and give us less time to do user studies. There can be some variations in the sprint length if the team sees workload for sprints being off or scrum master sees a benefit in changing it.

For each sprint we will have planning, review and retrospective meeting. Sprint planning meetings will be used to discuss what the team would like to achieve using user studies for the upcoming sprint. There will be held planning poker sessions for estimating user stories to work with in the upcoming sprint. Sprint review meeting will be used to present what each team member has done in sprint and will have status update ready for the meeting with product owner. Sprint retrospective meeting will be used to discuss what went well in the sprint, what could be improved and also discuss estimate precision.

4.2 Plan for status meetings and decision

After each sprint group should have a meeting with Frantz, Christopher (product owner) every Monday at 13.15. In this meeting product owner will give his input on the update which will be used as user stories. We will also discuss the next sprint with product owner. Group should have a meeting with Kolloen, Oivind (supervisor) every Monday at 09:00. Group meetings for code review, discussing internal decisions and planning for meeting with supervisor and product owner is every Friday from 14:00 to 16:00. Sprint planning meeting will be held after the meeting with Frantz, Christopher (product owner).

5 Organization of quality control

The group is expected to have high standards for code quality. All members should follow the official guidelines for Golang style and code review. The w3schools style guides should be used for HTML5 and JavaScript. The CSS-tricks style guides should be used for SASS.

Each member will document and create tests for the functionality they add. Additional documentation is expected for API endpoints such as http GET and POST requests.

The group code review should be done during the Friday meetings and for every pull request. User testing will be done on a regular basis.

5.1 Configuration management

The main focus when setting up the development environment and use of technologies is to allow for seamless connecting between code, workflow and documentation. Considering the agile development method, the tools

must facilitate communication and actors with multiple roles. The agile development also calls for user studies and testing, this makes DevOps particularly advantageous.

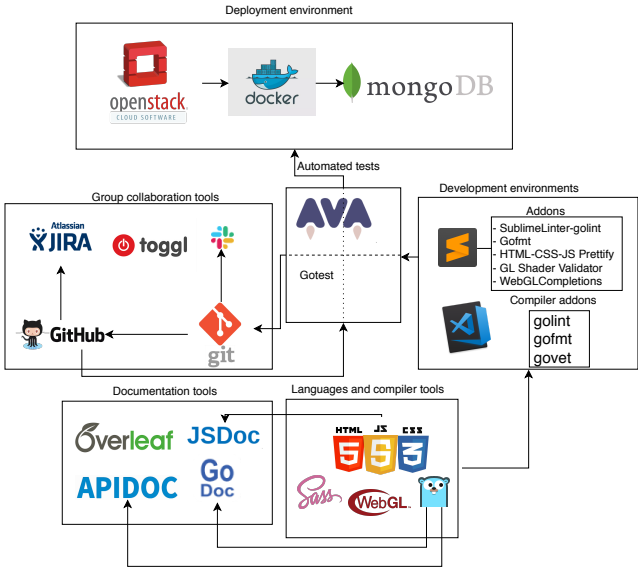

Figure 1: Tools to be used in the project

As shown in figure 1, Openstack with Jenkins will be used for rapid deployment and Infrastructure as Code to seamlessly document environment configuration. Docker will allow for further environment documentation and release on foreign systems. Jira with Github integration will be used to manage tasks. The Jira workflow is set to include issue status as "To Do", "In Progress", "Pending Review", "In Review" and "Done". The stories are estimated by the development team at inception and before a sprint using agile poker. Work is planned through Slack and work hours are logged using Toggl. Editors and IDEs choice is up to the individual but linters and formatters are required to validate a push to git. Languages are chosen to reflect the standards or allow good collaboration and high readability. Go was chosen in particular for its strict syntax and integration with tools such as GoDoc and gofmt as well as a built in testing framework. For seamless connection between code and documentation GoDoc, APIDOC and JSDoc were chosen for their respective language and use case. APIDOC will be used extensively to generate a comprehensive and thorough documentation on the REST API. Insufficient documentation will not be accepted by merge request and feedback on merge requests will ensure that all users follow the same standards and can discuss when different opinions on the matter surfaces. Additional documentation, like wiki pages for structural overview and meeting reports for covering decisions and development processes will be created using LaTeX and Overleaf for collaboration.

The documentation will hence tightly follow the development process and allow the developer to document their code whilst writing it, the infrastructure will be documented as a side effect of the DevOps automation and the development process will be documented through references to issues.

Automated testing will be done using tools like gotest and AVA to ensure the code on Github is functional and validated before deployment on OpenStack.

5.2 Risk analysis

5.2.1 Identification and project risk analysis

The table 2 contains possible risks based on [1] and compared with the system requirements. The probabilities and effects are estimates done by the core team. The priority is based on the minimal Manhattan distance from top right corner of table 3.

Id	Risk Type	Possible Risk	Priority	Probability	Effect
1	Organization	Losing important data	8	Low	Significant
2	Technology	Technology components aren't scalable	4	Medium	Severe
3	Technology	Insufficient security for the system (Information security)	5	Medium	Significant
4	Technology	Application instability	6	Low	Severe
5	Organization	Team members are unavailable for longer period of time	7	Medium	Significant
6	Requirement	Discovering problems in requirements at delivery	1	High	Severe
7	Estimate	Project size is under estimated	3	High	Significant
8	Organization	Learning curves lead to delays	2	Very High	Moderate
9	Technology	Integration testing environments aren't available	10	Low	Minor
10	Organization	Fail to follow software methodology	9	Medium	Minimal

Table 2: Overview of risks

	Minimal	Minor	Moderate	Significant	Severe
Very High			8		
High				7	6
Medium	10			3, 5	2
Low		9		1	4
Very Low					

Table 3: Risk analysis form

5.2.2 Risk mitigation strategy

Table 4 shows the mitigation strategy for the project risks with highest priority and should give an overview of mitigations that should help with most of the identified risks, as several can be mitigated with similar strategies.

Priority	Risk	Mitigation
1	Discovering problems in requirements at delivery	• Use an agile development process • Perform regular user testing
2	Learning curve lead to delay	• When a new technology is to be use a workshop for exploring it will be held
3	Project size is under estimate	• Use an agile development process • Have regular meetings with product owner
4	Technology components are not scalable	• Use IaC and code for modularity
5	Insufficient security for the system (Information security)	• Test coverage of all important functions • Use of static analysis tools
6	Application instability	• Unit test coverage of all important functions • Use DevOps with live server throughout the development
7	Team members are unavailable for longer period of time	• Focus on reviewing git pull requests well • Good communication amongst developers

Table 4: Risk mitigation

6 Development plan

6.1 GANTT - diagram

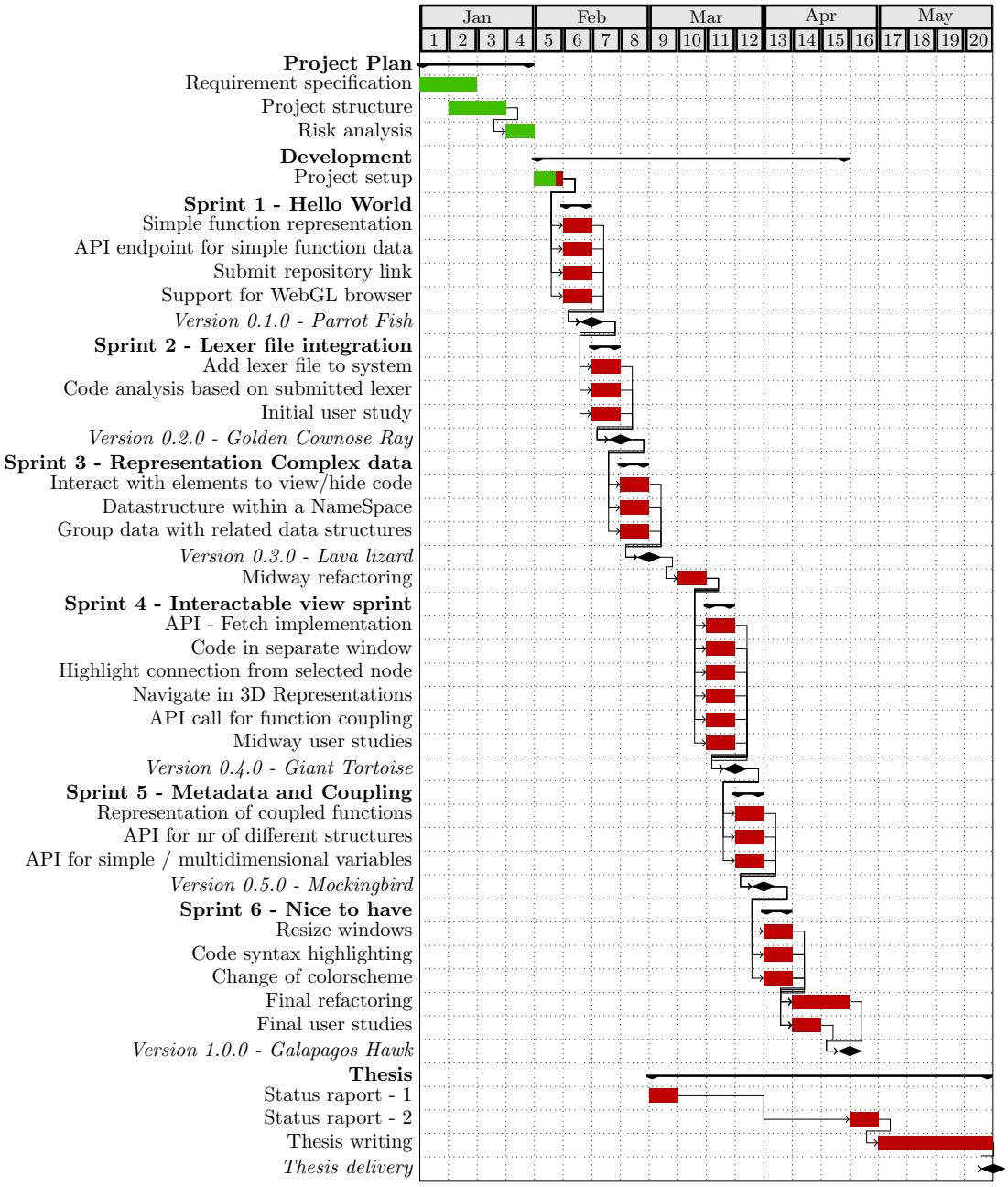

Figure 2: Gantt diagram

The diagram shows the timeline from left to right of the entire project. Green is used to show competed progress while red is used for what is not completed.

Summary	Issue key	Issue Type	Sprint	Story Points
As a developer i would like the API endpoint extended to give enough to represent simple and multidimentional variables.	COD-7	Story	Metadata and coupling sprint	8
As a developer i would like the API to include request for implementation based on function/class/etc name or start/end markers	COD-25	Story	Interactable view sprint	3
As a developer i would like an API endpoint giving metadata about the project.	COD-13	Epic		
As a developer i would like the API endpoint extended to give information about what functions call what.	COD-9	Story	Interactable view sprint	3
As a user I would like to see what functions are tightly coupled in the 3D representation	COD-8	Story	Metadata and coupling sprint	5
As a user i would like to navigate the 3d representation.	COD-10	Story	Interactable view sprint	8
As a developer i would like the API endpoint to include comment coverage and test coverage	COD-15	Epic		
As a user i would like to submitt a git repository link for the system.	COD-6	Story	Hello world sprint	3
As a user i would like to view or hide the implementation of a function or class by clicking it.	COD-11	Story	Represent complex data sprint	5
As a developer i want an API endpoint that gives me enough to represent a basic function.	COD-4	Story	Hello world sprint	13
As a user i would like to see a representation of a simple function	COD-3	Story	Hello world sprint	8
As a user I want to submit hello world repository and visualize my code in 3D	COD-2	Epic		
As a developer i would like coding language for analysis to be based on provided lexer file.	COD-24	Story	Lexer file integration sprint	5
As a user i would like to submitt a lexer file for defining language syntax	COD-23	Story	Lexer file integration sprint	13
As a user i would like to change the colorscheme.	COD-18	Story	User experience sprint	3
As a user i would like code to be displayed with syntax highlighting	COD-22	Story	User experience sprint	5
As a user i would like to resize windows	COD-20	Story	User experience sprint	8
As a user i would like to highlight connections from selected node.	COD-12	Story	Interactable view sprint	3
As a user i would like to display code in a seperate window	COD-21	Story	Interactable view sprint	3
As a user i would like to see test coverage and comment coverage for Golang	COD-26	Story		
As a user i would like the data structures in the 3D visualization to group together with related structures	COD-19	Story	Represent complex data sprint	8
As a user i would like to see what namespace data structures are in.	COD-17	Story	Represent complex data sprint	3
As a user i would like to be notified if my browser does not support WebGL or this application	COD-5	Story	Hello world sprint	2
				112

Table 6: Initial story estimates

6.2 Comments on the schedule

The schedule is flexible and should be changed whenever the priorities change. It might be altered depending on the user studies, product owner changing requirements or design problems. The intent is to use scrum where sprint lengths vary and incomplete stories can be put in the backlog or added to the next sprint. The project features are expected to be defined as user stories refined from user studies results. The user studies take precedence over the developers initial understanding of the users needs and priorities.

6.3 Time, Resource and Cost overview

As of now there are no expected costs. Our resources are limited to workhours and 3 employees only.

Member	Allocated workhours
Kent Wincent Holt - 473209	400 h
Eldar Hauge Torkelsen - 473180	400 h
Zohaib Butt - 473219	400 h
	1200 h

Table 5: Total workhours

The initial story estimates in table 6 resulted in 112 story points. The allocated time of 1200 workhours results in 10 hours per story point. The estimates will be re-estimated on the beginning of relevant sprint and the stories will change.

References

[1] Khaloo P, Maghoumi M, II EMT, Bettner D, Jr JJL. Code Park: A New 3D Code Visualization Tool. CoRR. 2017;abs/1708.02174. Available from: http://arxiv.org/abs/1708.02174.

[2] Wettel R. Software systems as cities. Università della Svizzera italiana; 2010.

[3] UNION EORFDE. Lov om behandling av personopplysninger (personopplysningsloven) EUROPAPARLAMENTS- OG RDSFORORDNING (EU) 2016/679 av 27. april 2016 om vern av fy-

siske personer i forbindelse med behandling av personopplysninger og om fri utveksling av slike opplysninger samt om oppheving av direktiv 95/46/EF (generell personvernforordning) [GDPR]. https://lovdata.no; 2016.
https://lovdata.no/dokument/NL/lov/2018-06-15-38/KAPITTEL_gdpr#KAPITTEL_gdpr.

[4] Krueger CW. Software Reuse. ACM Comput Surv. 1992 Jun;24(2):131–183. Available from: http://doi.acm.org/10.1145/130844.130856.

[5] Mileva YM, Dallmeier V, Burger M, Zeller A. Mining trends of library usage. In: Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops. ACM; 2009. p. 57–62.

[6] Nguyen-Duc A. The impact of software complexity on cost and quality - A comparative analysis between Open source and proprietary software. CoRR. 2017;abs/1712.00675. Available from: http://arxiv.org/abs/1712.00675.

[7] Spinellis D. Reading, writing, and code. Queue. 2003;1(7):84.

[8] Westland JC. The cost of errors in software development: evidence from industry. Journal of Systems and Software. 2002;62(1):1–9.